5-4-2008

Implementing Bluetooth Support In Wifi-Based Mobile Ad-Hoc Networks

Christopher Dragga
Macalester College

Honors Paper

Macalester College

Spring 2008

Title: Implementing Bluetooth Support in Wifi-Based Mobile Ad-Hoc Networks

Author: Christopher Dragga

SUBMISSION OF HONORS PROJECTS

Please read this document carefully before signing. If you have questions about any of these permissions, please contact Janet Sietmann in the Library.

Title of Honors Project: Implementing Bluetooth Support in WiFi-Based Mobile Ad-hoc Networks

Author's Name: (Last name, first name) Dragga, Christopher

The library provides access to your Honors Project in several ways:

- The library makes each Honors Project available to members of the Macalester College community and the general public on site during regular library hours.
- Using the latest technology, we make preservation copies of each Honors Project in both digital and microfilm formats.
- Every Honors Project is cataloged and recorded in CLICnet (library consortium OPAC) and in OCLC, the largest bibliographic database in the world.
- **To better serve the scholarly community,** a digital copy of your Honors Project will be made available via the Digital Commons @ Macalester (digitalcommons.macalester.edu.

The DigitalCommons@Macalester is our web based institutional repository for digital content produced by Macalester faculty, students, and staff. By placing your projects in the Digital Commons, all materials are searchable via Google Scholar and other search engines. Materials that are located in the Digital Commons are freely accessible to the world; however, your copyright protects against unauthorized use of the content. Although you have certain rights and privileges with your copyright, there are also responsibilities. Please review the following statements and identify that you have read them by signing below. Some departments may choose to protect the work of their Honors students because of continuing research. .In these cases the project is still posted on the repository, but content can only be accessed by individuals who are located on campus.

The original signed copy of this form will be bound with the print copy of the Honors Project. The microfilm copy will also include a copy of this form. Notice that this form exists will be included in the Digital Commons version.

I **agree** to make my Honors Project available to the Macalester College community and to the larger scholarly community via the Digital Commons@Macalester or its successor technology.

Signed Christopher C. Dragga

OR

I **do not want** my Honors Project available to the larger scholarly community. I want my Honors Project available only in the library, **NOT** for interlibrary loan purposes, and **NOT** through the Macalester College Digital Commons or its successor technology.

Signed _____

NOTICE OF ORIGINAL WORK AND USE OF COPYRIGHT PROTECTED MATERIALS:

If your work includes images that are not original works by you, you must include permissions from original content provider or the images will not be included in the electronic copy. If your work includes discs with music, data sets, or other accompanying material that is not original work by you, the same copyright stipulations apply. If your work includes interviews, you must include a statement that you have the permission from the interviewees to make their interviews public.

BY SIGNING THIS FORM, I ACKNOWLEDGE THAT ALL WORK CONTAINED IN THIS PAPER IS ORIGINAL WORK BY ME OR INCLUDES APPROPRIATE CITATIONS AND/OR PERMISSIONS WHEN CITING OR INCLUDING EXCERPTS OF WORK(S) BY OTHERS.

All students must sign here.

Signature: Christopher C. Dragga **Date:** 5/5/08

Printed Name: Christopher C. Dragga

Implementing Bluetooth Support in Wifi-Based Mobile Ad-Hoc Networks

Christopher Dragga

May 4, 2008

Contents

Abstract

Mobile ad-hoc networks (MANETs) provide a useful means of connecting computers in unusual situations, such as search and rescue. However, they ignore those small, highly mobile devices that only support Bluetooth, a low-range, low-bandwidth Wifi alternative that consumes significantly less power. Allowing these devices to connect to Wifi MANETs could permit a variety of applications, from text messaging to VOIP to parallel processing. Bluetooth features several unusual characteristics that could make this difficult, though. In this project, I implemented this kind of integration and analyzed its success through physical testing and models both analytical and simulated.

1 Introduction

In many ways, the Bluetooth communications protocol seems underused today. This wireless technology was originally developed as a replacement for cable connections, but has since gained significantly more functionality, including the ability to act as a replacement for the more prevalent IEEE 802.11 wifi networks generally used today. It has a number of features that would recommend it over wifi, especially for small and portable devices, like cell phones, handheld computers, and even laptops. Chiefest among these is its power consumption, which is significantly lower than that of wifi. It also supports a variety of services that allow the user to tailor its use specifically to his or her needs.

Most consumers today are only familiar with Bluetooth headsets for cell phones, Bluetooth printers, and other devices that make simple use of the technology. A closer look at Bluetooth suggests reasons why the more advanced applications of the technology are not more common. Though Bluetooth is present on many devices, its implementation is often inconsistent, especially with the services it supports; the standard Bluetooth device that comes in most Macintosh computers, for instance, does not support the service that emulates wifi [2]. In addition, there are a number of restricting limitations on the medium itself, including lower bandwidth, lower range, and behavior that differs drastically from wifi in a number of important ways, such as the inability to broadcast messages to all devices in the area.

On a similar note, computing is becoming increasingly more pervasive today. Cellphones are becoming more complex, and handheld devices that basically function as small computers, like the Nokia N800, are starting to appear on the market. Currently, though, many of these devices are underused, and their usefulness has yet to fully materialize. The aforementioned N800 is a prime example of this: the device is quite powerful, but little software exists for it.

This research project looks into using both of these underused technologies by integrating Bluetooth communications into wifi-based mobile ad-hoc network protocols. These protocols are designed for large numbers of mobile devices without any centralized structure, so that devices may enter and leave at will. This format complements the nature of handheld devices, which often only support Bluetooth (although wifi support is increasingly common) and which are designed for mobility.

There are a number of potential applications for such networks. Ideally, they could supplant cell phone calls with VOIP carried by Bluetooth, as suggested in [22] which would allow for calls to be made in areas with no cell phone reception

and could thus be used to avoid consuming cell phone plan minutes. From a less ambitious standpoint, these networks could be used to allow phones access to the Internet without having to pay for cellular Internet. At the very least, instant messaging could be done over these networks, avoiding the potentially costly use of text messages. From a more research-oriented standpoint, it would be possible to use a network such as this for a distributed computing framework, making it easier to leverage the admittedly limited, though still potentially useful, processing power of Bluetooth-only devices.

In this paper, I will first provide background on both mobile ad-hoc networks and Bluetooth in section 2, and look at some of the difficulties created when attempting to combine the two, in section 3. Then, in order to get a better idea of how a mobile ad-hoc network that integrates Bluetooth might perform, I will examine in section 5 several mathematical models of mobile ad-hoc networks and try to apply their results to the protocol described here. In addition, I will look at the results of simulation models and papers that deal with the proper ways to perform them. Section 4 describes the steps I took to actually physically implement this network, so as to get some idea of its feasibility. I then used the statistics I gathered from my physical implementation and its testing in section 6.2 to simulate a Bluetooth-integrating MANET, the results of which I detail in section 6.3. For the simulation, I will look principally at the time the devices spent connected, as that should give some indicator of how smoothly the network will perform in various circumstances, which in turn will help determine which of the above applications will be viable under this framework.

2 Background

2.1 Mobile Ad-hoc Networks

Mobile Ad-hoc Networks, or MANETs, have been the subject of a sizeable body of research in recent years. As mentioned earlier, these are networks designed to support large numbers of mobile computers, without any kind of centralized servers or routers [14].

In 1999, S. Corson and J. Macker published RFC 2501, a document that describes the principles behind MANETs in brief detail. While this was not the first work on the subject, as many of the ideas that MANETs employ have existed in various forms since the 1970s, it is the first RFC (RFCs are standards documents published by the Internet Engineering Task Force) published on the matter and

stands as a useful reference. Some of the primary characteristics of MANETs it describes are fairly intuitive, such as the changing nature of their topologies, but most are not readily apparent. MANETs often need to monitor their energy consumption, as mobile devices often use batteries, and bandwidth often fluctuates greatly, due to both the movement of the devices and the limitations of wireless networking. Security also poses a problem, again because of the nature of wireless networking. Note that there is no mention of the wireless protocol used; hypothetically, MANETs can work over any means of wireless communication, though most MANET implementations assume a medium similar to the IEEE 802.11 WLAN interface.

The RFC also discusses a number of potential metrics that can be used to judge the performance of MANET routing protocols, including throughput, delay, and the time it takes to acquire routes from one device to another. Another metric is the "efficiency" with which the protocol operates–how much network traffic is required to transmit messages and to keep the network running smoothly. In addition, the RFC stresses the consideration of the characteristics of the network itself when measuring performance, rather than merely examining the protocol in a vacuum. These characteristics include the amount of movement within the network and the frequency thereof (if movement is constant, or if it starts and stops), how spread out the network is, and how fast the links can transmit data. All of these will directly affect a MANET's performance, and some protocols may be affected differently than others. One protocol might perform well in networks with little movement but poorly when significant movement and unreliability are introduced. In contrast, a different protocol might perform more slowly than the first under ideal circumstances but handle better under difficult conditions.

Research into MANETs has yielded a large number of protocols, though most of these have only been proposed and not implemented. Of those that have implementations, three stand out as particularly important: the Optimized Link State Routing protocol (OLSR), the Dynamic Source Routing protocol (DSR), and Ad Hoc On Demand Distance Vector (AODV) routing. Each of these protocols has at least four different implementations [37], and each has an RFC that explains it in detail [18].

The first of these, OLSR, is an example of a proactive protocol. This means that it discovers routes actively, regularly flooding the network with control messages, which provide information about neighbors and link strength. Ordinarily, this could clog the network with excess traffic; however, the protocol avoids this problem by designating certain nodes as "multipoint relays" (MPRs). These MPRs are the only nodes that actually forward the control messages they receive,

which reduces the traffic by a significant amount [12].

AODV and DSR, on the other hand, are both reactive protocols, meaning that they discover routes as necessary, rather than attempting to keep the network topology constantly up to date. Ideally, this strategy reduces network overhead and improves the network's ability to accomodate large numbers of nodes [30]. As in OLSR, route requests propagate through flooding, though there are no equivalents to MPRs. For both protocols, a node sends out route requests when it generates a message whose destination is not in the node's routing table [30, 20]. In addition, AODV will send out a route request when a node receives a message with a destination for which it lacks a route [30]. This difference arises through the way each protocol specifies the routes for their packets. DSR attaches the entire route from source to destination to the packet, whereas AODV specifies only the source and destination nodes [30, 20]. Partially because of this, the two protocols perform quite differently: AODV generates better throughput and less delay than DSR in situations where devices are moving quickly and there are numerous nodes, while DSR tends to do better in those metrics than AODV in circumstances with less mobility and fewer nodes [31].

In addition to these three, other routing protocols have been proposed. Pei, Gerla, and Chen have proposed "Fisheye State Routing", a protocol that attempts to reduce the overhead caused by maintaining a full network topology by only updating nearby nodes in the network graph with any frequency [29]. Kyasanur and Vaidya propose a protocol that takes advantage of both multiple channels and multiple wireless devices on a single device [25]. Jipping and Lewandowski have proposed the HAND network model, which includes a light MANET layer designed to work on handheld devices [19].

Although few of these protocols have working implementations, most of them have been tested using simulations. Even major protocols like AODV and OLSR are often simulated to test for large-scale performance, as the cost of procuring the necessary equipment often precludes a real-world test. Currently, the most popular simulator is ns-2, though this apparently experiences problems with scalability. Kargl and Schoch argue that the JiST/SWANS simulator actually functions better on the whole, though it has yet to come into wide use [23].

Most researchers regard their simulated results with a high level of confidence, but some have raised questions over the accuracy of these statistics. In particular, Kiess and Mauve examined implementations of two MANETs, including AODV, in 2002 and found that a number of problems arose in keeping connectivity that never appeared in simulations. In particular, they found poor signals that occasionally gained high strength to be a major problem; the protocols would assume

these links would remain at high strength, and then would waste a fair amount of time repairing the network when transmissions timed out after the links returned to their usual low connectivity [11].

Simulation is not the only form of testing that occurs, however. Toh et al describe the methods they used to test a new MANET protocol on a college campus, giving them information on throughput and end-to-end delay, among other statistics [35]. Some researchers also model MANETs. Khalil et al use an epidemic model to determine the optimal number of nodes for information to fully flood a network, determining the density to be about 620 nodes/km^2 [24]. In addition, Cai and Eun use random-walk models to verify a power-law distribution for the time between meetings of mobile nodes in an unbounded area, and an exponential distribution for a bounded area [8].

Finally, some contend the idea that MANETs are a particularly efficient way to coordinate mobile traffic. Mahmud et al argue that MANETs are actually quite inferior in most applications to mesh networks, which feature a number of static routers in addition to the mobile nodes [27]. While they may have a point, mesh networks lack some of the versatility of MANETs since they require a fixed architecture and thus cannot be spontaneously deployed. Thus, research into MANETs continues today.

2.2 Bluetooth

Bluetooth exists as one of the principal alternatives to the IEEE 802.11 wireless interface. Like the 802.11 interface, Bluetooth employs the 2.4 GHz band, leaving it open to interference from other devices. To circumvent this, Bluetooth devices perform numerous frequency hops when operating [17]. Bluetooth's bandwidth rate is limited to 1 Mbps and most devices have a range of ten meters. Compared to 802.11, these capabilities are quite limited, but Bluetooth is not intended to be a direct competitor to this interface. Instead, it is designed to facilitate "Personal Area Networks" (PANs) of electronic devices, many of which may be small and battery powered. The lower bandwidth and range of Bluetooth actually facilitate this, since they allow for significantly lower power consumption [17].

To provide easy interfacing between applications, Bluetooth devices employ different service profiles, which provide different methods of communication between devices [17]. These include SPP, the serial port profile, which emulates a serial connection between two devices; FTP, the standard Internet file transfer protocol; and OPP, or object push profile, which is another means to transfer files between Bluetooth devices[5]. These profiles operate on top of the several protocols

that Bluetooth supports [17]. The most notable of these are the Service Discovery Protocol (SDP), which allows Bluetooth to discover the services present on other devices; BNEP, which allows for transport of IP and other networking protocols; and RFCOMM, which provides serial port emulation [5].

Unlike 802.11, which is a broadcast medium, Bluetooth focuses on making specific connections between devices. When connecting to each other, Bluetooth devices form piconets of up to eight devices, wherein one is a master and the others are slaves. The master coordinates communication among the slaves and determines the series of frequency hops [17].

The small number of Bluetooth devices that can connect to each other could prove a limiting factor. However, according to the Bluetooth specification, a device can be a master or slave in one piconet and a slave in another, forming a so-called "scatternet". Unfortunately, very few Bluetooth devices actually implement this feature owing to difficulty in devising efficient scheduling algorithms for such an arrangement [17]. A decently large body of research exists describing potential algorithms; Racz et al, for instance, devised a method that coordinates scheduling between piconets via a pseudo-randomly generated sequence of "meeting points" between nodes in different networks [32]. They find that the algorithm performs almost as well as the ideal scheduling algorithm; unfortunately, at no point do they give quantitative numbers about performance, such as the average delay. Misic and Misic actually do analyze the delay in a two piconet scatternet, finding that it can reach into "the range of seconds", though they hasten to note that such delay is acceptable for certain applications [28].

Much of the research into Bluetooth at this point seems to be theoretical. Kapoor et al, authors of [21], look at support for transporting multimedia over Bluetooth, comparing the results to 802.11. Surprisingly, Bluetooth performs quite well, partially because its connection-oriented nature cuts down on interference. However, the article seems to assume that scatternets exist, since it discusses additional Bluetooth nodes increasing network capacity, which can only occur if piconets can interact with each other. Thus, the practical results of this study are dubious.

Similarly, a paper that describes a system integrating Bluetooth and 802.11, much like this project, relies upon devices that combine both systems, as well as the ability to sense signal strength, which the Bluetooth specification does not support [13]. Another paper, [10], describes means of interacting between IP access points and Bluetooth devices and claims that its proposals can be implemented with current technology. However, it appears to rely upon scatternets, which do not function for most Bluetooth devices.

3 Difficulties with Bluetooth Based MANETs

Now that there has been a background on both the Bluetooth medium and MANET protocols established, it is worth considering how the two will interact, given the details of their functionality. This will directly inform the best way to implement a MANET with Bluetooth.

The inability to form scatternets stands as the largest barrier toward successful MANET implementations in Bluetooth. Certainly with scatternets, Bluetooth MANETs are entirely possible; [32] uses AODV as the routing protocol when testing the performance of its scatternet algorithm. However, [22] describes a potential Bluetooth-based MANET protocol, but notes that it is currently impossible to implement on today's hardware because no common devices implement scatternets.

Even allowing for scatternets, though, Bluetooth has several quirks that need to be considered when designing any networking system that employs it. Firstly, the connection-oriented nature of Bluetooth forces a new connection for every data transfer and prohibits any kind of broadcasting of messages. Secondly, device discovery and connection setup are both relatively lengthy processes, requiring careful consideration of how connecting is handled. Because of this, straightforward "ports" of MANET protocols to Bluetooth will face extensive delay times in communication [22]. Also, though not directly related to Bluetooth, many devices that run Bluetooth have relatively low memory and processor capabilities. Thus, they may have difficulty running some of the more intensive protocols, such as DSR, which requires the sending device to know the entire route of the packet [19].

Given these limitations, the most efficient method for integrating 802.11 and Bluetooth into MANETs would be to use devices that have access to both interfaces to act as network access points (NAPs) to the 802.11 network, and to allow Bluetooth-only devices to communicate solely with these. Bluetooth directly supports this use of the technology through the PAN service profile [4], which allows for connection to ethernet services through NAPs and assigns IP addresses to the Bluetooth devices, among other things. Unfortunately, support of this service is spotty; Mac OS X, for instance, does not provide it [2], though Linux does [6]. Using it would thus lead to problems with porting the protocol to different operating systems, though it would faciliate the coding.

The alternative to using PAN would be to employ SPP, a basic service which would allow the packet data to be sent across a serial-like connection. This would provide the necessary data transfer capabilities, although assigning a valid IP ad-

dress to a Bluetooth-only device could be tricky. However, data transfer over SPP is limited to 128 kbps [5], which could restrict its real-time applications (voice transmissions, for instance, could become difficult). Alternatively, OPP (Object Push Profile) could be used, though the operation of this is often quite clumsy and may involve prompting the user to accept file transmissions on certain devices. OPP also lies atop SPP [5], so devices that support one are likely to support the other.

In addition to Bluetooth considerations, one must also decide upon a MANET protocol to employ. Because this project's goal is to integrate Bluetooth into 802.11 MANETs, writing a completely new MANET protocol makes little sense, given time constraints. Furthermore, few implementations of MANET protocols exist, as mentioned before, limiting the choice to one of DSR, OLSR, or AODV. Because Bluetooth devices often have fairly tight memory constraints, DSR can be eliminated almost immediately, since each node needs to have complete knowledge of the routes its packets travel. AODV and OLSR, while they do require the nodes to keep a routing table, do not require the entire route to go into the packet header, allowing Bluetooth-only devices to have only a rudimentary knowledge of the network topology. In addition, the lengthy Bluetooth device discovery process poses some problems. Because OLSR is an active protocol, it constantly queries its nodes to ensure that connections hold stable, which could presumably allow for fast discovery of a Bluetooth link break. In contrast, because AODV is reactive, it will likely not discover a Bluetooth link break until it tries to send a packet, at which point it will likely take a number of seconds to rediscover the device and reestablish a route. While a more active behavior for Bluetooth nodes could be programmed into AODV, it is probably best to try to keep the protocol operating under its standard set of principles.

Given these considerations, I chose to implement this using the PAN service and AODV, due to the relative ease which these provide. Thus, the architecture of the protocol will have each Bluetooth only device designated as a PANU, and each bluetooth-wifi gateway defined as a NAP. I describe this implementation in the following section.

4 Implementation

Because most Bluetooth research seems to exist only in theory, I decided to implement the Bluetooth-wifi MANET integration, both to see whether this architecture would work and to investigate the difficulties of programming for Bluetooth de-

vices. This entailed choosing a MANET protocol and then determining a way to have Bluetooth devices automatically connect to each other and transfer data. From there, I had to figure out a way to have the MANET protocol use the Bluetooth interface in addition to the 802.11 interface.

For the MANET portion of this project, I decided to use AODV. While not ideal for the small devices likely to run only Bluetooth, it still seemed better suited than the other mainstream alternatives, OLSR and DSR. Because it is a proactive protocol, OLSR requires nodes to constantly send routing messages throughout the network, which could potentially clog the limited bandwidth of Bluetooth. AODV only requires route discovery messages when necessary, which would cut down on overhead. DSR, while reactive like AODV, requires sending nodes to have complete information about routes and requires the message to contain the entire route, whereas for AODV, the message need only contain the beginning and destination and the sending node need only know the next hop along the path. Because of the nature of the Bluetooth component, Bluetooth only nodes will never forward messages and thus can save storage space by only storing the next hops. Similarly, the decreased message size will help save bandwidth.

The specific AODV implementation I chose to use was AODV-UU, due to its standards compliance and its ready availability. This implementation is written for Linux, so to program the Bluetooth connectivity component, I used Bluez, Linux's default Bluetooth library.

Bluetooth, and thus, Bluez, provide a number of different services which allow for connections between devices. As mentioned in section 3, I chose to use the PAN service, which, when configured, allows Bluetooth to work analogously to a standard network device. The standard Bluez distribution provides a daemon, pand, that facilitates PAN connections. While the listening functions on this work excellently, allowing for NAP nodes (those with both Bluetooth and wifi access, allowing the pure Bluetooth nodes access to the broader network) to listen indefinitely for connections and run configuration scripts upon connection, the connection functions proved inadequate for my purposes. The daemon allegedly supports periodic polling of nearby nodes to attempt to connect, but I was unable to get this to work properly. Because of this, and also to experiment with the usability of the Bluez API, I decided to create my own program to connect to NAP nodes.

Rather than use direct C calls to the operating system, Bluez relies on DBUS, Linux's intra-system message passing system, to do most of its work. Essentially, the utility that activates Bluetooth, hcid, sets up a number of objects to which the user can send DBUS messages to make them perform most Bluetooth tasks,

such as device discovery and establishing connections. This allows for portability between languages–the languages need only an interface to communicate over DBUS–and, hypothetically, simplifies Bluetooth programming for the end-user. Unfortunately, the documentation for Bluez is fairly sparse, consisting principally of API documentation for the DBUS objects. Examples of code, especially for C, the language in which I worked, are rare, and the GLIB DBUS API features arcane and convoluted syntax, leading to a steep learning curve.

To access the Bluetooth API, one must first establish a connection to the DBUS system bus (not the session bus, which is also an option). From here, one can then create proxies for the different Bluez objects and call methods on the proxies, which then invoke the methods on the real objects and return the results. The procedure for doing this is fairly straightforward: in the call, one must specify the function called, the parameters and parameter types, and then the return types. The only real complication occurs with determining what types to use. The API appears to have been written mainly for use with the Python DBUS libraries, and specifies return types like arrays of strings and dictionaries. A perusal of the examples on the Bluez wiki [6] reveals that string arrays correspond to char ** in C, while I believe dictionaries are returned as the type *GHashTable.

An example of a typical DBUS call appears below. In this case, the call is to the ListAdapters method, which provides a list of the Bluetooth adapters available. The syntax, while cumbersome, is at least consistent.

```
if (!dbus_g_proxy_call(proxy, "ListAdapters",
     &error, G_TYPE_INVALID,
     G_TYPE_STRV, &name_list, G_TYPE_INVALID)){
           g_printerr("Error:  %s\n", error->message);
           g_error_free(error);
           exit(1);
     }
```

The syntax becomes significantly more complicated when callback functions are necessary, as is the case with device discovery and connection establishment. After a method with a callback is called, the DBUS object will then emit a signal, which will result in a function from the original program being called. In order to perform this, one must register a signal marshaller for the signal's parameters, register the signal, and link the signal to the callback function. GLIB provides a number of single parameter signal marshallers, but in order to create a marshaller for a signal with multiple parameters, one must run the glib-genmarshal

program, which creates both source and header files for a marshaller with parameters specified when starting the program, and then compile these files with the final program. The signature for the callback function must take as its first parameter the object that sends the signal, followed by the parameters for the signal and a void pointer to user-defined data specified when the callback is linked to the signal. The procedure on the whole is byzantine and far from intuitive; I determined the necessary callback signature more or less through experimentation and some extensive Google searching.

The following code provides an example of how to register a signal with its callback, as described above. In this case, this links the "Remote Device Found" signal with the callback function remote_device_found_handler. This signal is generated during device discovery, when the Bluetooth adapter detects another adapter within range. The signal calls the method with the address of the discovered Bluetooth device, the class of the discovered device (represented in an unsigned integer), and an int for the rssi value.

```
dbus_g_object_register_marshaller(
      g_cclosure_user_marshal_VOID__STRING_UINT_INT,
      G_TYPE_NONE, G_TYPE_STRING,
      G_TYPE_UINT,G_TYPE_INT,
      G_TYPE_INVALID);
dbus_g_proxy_add_signal(adapter, "RemoteDeviceFound",
      G_TYPE_STRING,
      G_TYPE_UINT, G_TYPE_INT);
dbus_g_proxy_connect_signal(adapter, "RemoteDeviceFound",
      G_CALLBACK(remote_device_found_handler),
      connection, NULL);
```

The actual procedure for establishing a PAN connection is fairly straightforward. All objects mentioned here are described in either the standard Bluez API or in the network service API.First, one uses the Adapter object to initiate device discovery. Then, once a device is found, assuming it has the necessary services (more on this later), one can actually establish a connection. This involves first using the Manager object to start the network service and to obtain a proxy object for it. Then, one uses the network service to create a connection proxy for the Bluetooth address of the discovered device. Finally, one invokes the "connect" method for the NAP service on the connection object. This method returns the network interface for the connection, which will be bnep0 if no other Bluetooth

connection exists. Assuming the device successfully connects, the connection will then send a "connected" signal. In the callback, it will then be safe to configure the interface with an IP address, which can be done by using execlp to run ifconfig from within the program. For this to function properly, the program must have been started with root privileges.

Once the device loses the connection, the connection object will generate a "disconnected" signal. In the case of my program, this causes the connection process to begin again, starting at device discovery.

In order for this procedure to work, the target of the connection must have the NAP service enabled; otherwise, the connection will fail. One way to determine this is to scan the services offered by the remote device and only attempt to connect if it offers the NAP service; unfortunately, service discovery may take several seconds, slowing connection time. Though I have not experimented much with this, it may be faster to automatically try to connect and abort if the service is not offered. This depends on one of the method calls reliably generating an error message, though, which I am not sure is the case. Currently, my program uses a service scan.

While the Bluez DBUS API works well for client connections, it does not work nearly as well for listening for connections. There appears to be no way to determine whether a connection has been established by a remote agent, making it next to useless for this purpose. In addition, running any program that employs the network services offered by the DBUS API renders pand non-functional. Finally, the documentation for this portion of the API is incomplete, and possibly incorrect in spots (at the time of writing).Thus, it is significantly more convenient to simply use pand to listen for connections on the NAP side. pand's functionality is documented at the Bluez pand HOW-TO, which provides a number of useful examples for setting up network connectivity [33].

Because AODV-UU allows the user to specify any network interface for its operation, the PANU can simply start it for the interface bnep0. Unfortunately, this does not work for NAP nodes, since they will have a separate interface for each connection made to them and they will need to communicate over the eth0 interface as well. I solved this problem using network bridging, directly as described in the BlueZ pand HOWTO. This involves creating a network bridge, pan0, with the eth0 interface automatically added to it. When invoking pand to listen, specify in the devup parameter a script that adds the bnep interface generated upon connection to the bridge. To run AODV-UU on the NAP after doing this, one then invokes aodvd, the AODV-UU daemon, for the pan0 interface.

Ultimately, my implementation proved reasonably successful. While several

quirks exist, as described in the performance testing, it should be possible to fix them, given time. My impression of the usability of the Bluez API is mixed, however. While the initial learning curve is immense, the coding becomes relatively straightforward, albeit wordy, after one learns the basics. The actual performance of the APIs as advertised seems shoddy, as demonstrated with my failure to use the server functions. However, the development team is updating the libraries at a rapid rate, and the issues may be fixed soon.

5 Modeling

Now that an implementation for a mixed Bluetooth-802.11 MANET exists, the next step is to investigate how well it works. Before engaging in direct simulation or testing, however, it helps to have an idea of how MANETs generally perform. This section will examine models, both analytical and simulation-based, to determine this. In addition to looking at general properties of MANETs, I will try to apply some of the analytical models to my own protocol.

5.1 Background

Modeling is of key importance to most MANET projects. Rigorously testing a MANET requires hundreds of systems distributed over a fairly broad area and numerous trials. Few organizations, especially in the research community, can muster the resources needed for such an effort. Thus, researchers usually turn to models to verify a MANET's performance.

In some cases, these models are analytical. This forces the possible scope of such models to be fairly narrow; if the models strive too hard toward realism, the analysis often becomes intractable. Thus, these models usually consider the general properties of MANETs, rather than those that derive from specific protocols, like the effect of multipoint relays in OLSR, for instance. Instead, certain models, such as [3] and [34], deal with the necessary conditions to keep a MANET connected, while others, such as [24] and [26], analyze the efficiency of information flow over the MANET. In addition, [13] presents a model for a situation similar to that with which I am dealing, albeit with a few notable differences.

Most models, however, are simulation-based. The majority of these use ns-2, though other simulators, such as JiST/SWANS, may work better [23]. These simulators go into a high level of detail in simulating different protocols; the code often resembles that of the actual MANET. A fair amount of discussion exists on how realistically simulations behave. Ref. [9] looks at the different mobility models one can use for the nodes in the network, and how the choice affects the results of the simulation, for instance, and [11] criticizes simulations for not adequately reflecting reality (in this case, the effects of weakening signal strength). Despite this, most papers that compare MANET protocols, like [31] and [7], rely entirely on simulation for their results.

5.2 Analytical Models

5.2.1 Connectivity and Transmission Range

Mobility in MANETs introduces the possibility that not all nodes may be connected to each other at a given time. Indeed, detection and repair of broken routes comprises a critical function of most MANET protocols, though these can be time-consuming operations. As such, analysis of the connectivity of a MANET can provide helpful insight into the overall efficiency of a MANET. This is especially important for Bluetooth, since its range is very limited compared to that of 802.11 wireless.

One of the most useful models I have encountered thus far is the subject of [3]. This paper analyzes the transmission range necessary to achieve different levels of connectedness within a given MANET using graph theory. Graphs consist of collections of nodes connected by links, or edges. When modeling a network, the nodes represent individual systems–in this case, PANUs, NAPs, or devices with only wifi access–and the links represent connections between them.

In performing analysis, [3] looks at two related metrics: minimum node degree across the network and k-connectivity. Node degree refers to the number of nodes which are linked to a specific node; thus, minimum node degree would be the minimum number of nodes attached to any specific node in the network. The direct analog to this in the actual MANET would be the minimum number of devices connected to any device in the network. Regarding the latter metric, a graph is connected when any node can be reached from any other node. A graph is k-connected when there exist at minimum k mutually independent paths from any given node to each other node. This corresponds to the number of routes that the data can take between devices in the MANET. In Figure 1, the minimum node degree is two, since node E is only connected to nodes D and B. The graph is also 2-connected, since there exist only two independent paths from node E to node A; while more paths exist, they all have to travel through either edge D-E or edge B-E.

To construct the model, the author assumes that the nodes in the network occur in a uniform random distribution across a two-dimensional plane. This is critical, since it allows the author to avoid actually modeling mobility; he assumes that the motion of the nodes will preserve the distribution, which means that change in position need not be accounted for. A pair of nodes can communicate if their transmission power, which decreases based on distance, is above a certain threshold; transmission range is based on this.

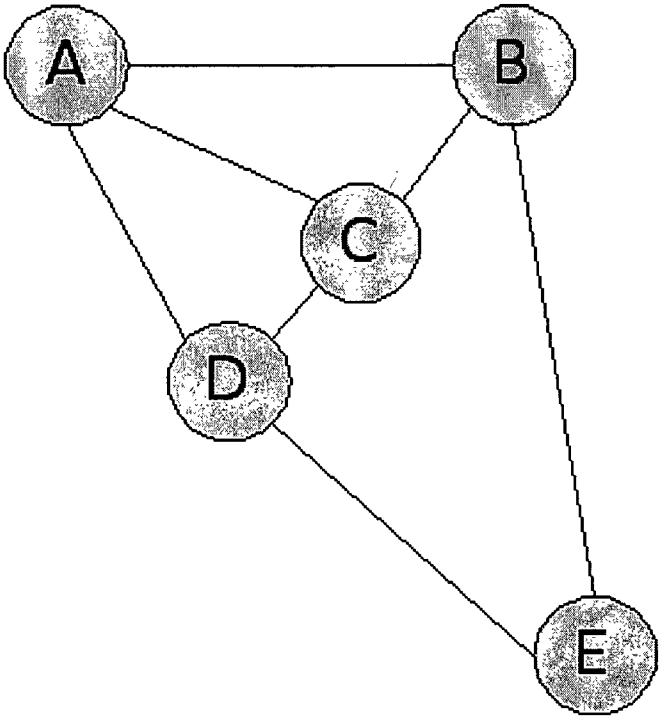

Figure 1: Example of minimum node degree and connectivity.

The analysis in the paper begins by simply attempting to determine the probability that a network will have no isolated nodes; that is, that each node will have at least one neighbor with which it can communicate. This does not, however, say anything about the connectivity of the network, as it is possible to have isolated "islands", where several nodes communicate with only each other but no other nodes in the network. To determine this probability, the paper looks at the nearest neighbor distance, the distance between a node and the closest node to it. For this, the paper considers the nodes to be a set of points arranged in a two-dimensional homogeneous Poisson point process. A Poisson point process is a means of modeling the layout of events, either in time or at specific locations, that occur at random intervals at an average rate based on a constant over a two dimensional plane, independent of previous points in the area[15]. As a result, the occurence of the points follows a Poisson distribution.

Preexisting work in spatial data analysis has shown that in this case, the probability density function for the nearest neighbor distance is $f(\xi) = 2\pi\rho\xi e^{-\rho\pi\xi^2}$, where ξ is the nearest neighbor distance greater than 0, and ρ is the node density in the initial area (the derivation is not given in the paper). From this function, the paper notes that the probability of the nearest neighbor being within transmission range is equal to the integral of this function from 0 to r_0, which evaluates to $1-e^{-\rho\pi r^2}$, where r_0 is the range. To find the probability that each node has at least one neighbor, one raises this to the n, to account for each node in the network:

$$P(d_{min} > 0) = \left(1 - e^{-\rho\pi r_0^2}\right)^n. \tag{1}$$

Here, d_{min} refers to the minimum degree of all nodes. From here, I solved this equation for node density necessary to achieve a probability p, yielding the following:

$$\rho = \frac{-\ln(1 - p^{\frac{1}{n}})}{\pi r_0^2}. \tag{2}$$

After this step, the paper expands the model to look at the probability of all nodes having minimum degree n. This is a more useful metric than merely having no isolated nodes, because a node that has multiple neighbors will not become disconnected from the network if one of its neighbors goes down. A node with only one neighbor in such a case will, however, become isolated. The math here is not a direct extension of the previous methods used, but the techniques are similar, as this portion also considers the nodes to be distributed in a Poisson point process.

First, the paper considers the one-dimensional case. To find the probability that a node has minimum degree n_0, it considers the probability that there are n_0

nodes within the transmission range of the node. This can be approximated with a Poisson distribution, producing the following formula where the range is r_0:

$$P(d = n_0) = \frac{(\rho 2r_0)^{n_0}}{(n_0!)} e^{-\rho 2r_0}, \tag{3}$$

the probability that one node has a degree of exactly n_0. The quantity $2r_0$ appears in the formula because the "area" covered on the one-dimensional line by the node's transmission is composed of the transmission range on each side of the node.

To arrive at the two-dimensional case, the paper changes lengths to areas in the Poisson distribution, yielding, after similar manipulations,

$$P(d = n_0) = \frac{(\rho \pi r_0^2)^{n_0}}{(n_0!)} e^{-\rho \pi r_0^2}. \tag{4}$$

Here, $2r_0$ changes to πr_0^2 to reflect that the area covered by the wireless transmission is now circular, rather than linear.

The final equation for the probability that all nodes have at least n neighbors is

$$P(d_{min} \geq n_0) = \left(1 - \sum_{N=0}^{n_0-1} \frac{(\rho \pi r_0^2)^N}{N!} e^{-\rho \pi r_0^2}\right)^n. \tag{5}$$

This can be obtained by noting that the probability that a node has at least degree n_0 is equal to one minus the sum of all probabilities that a node has a specific degree less than n_0, taken to the nth power, so that all nodes in the network are considered. This formula is significantly more difficult to use and calculate than the previous formula for the probability of no isolated nodes, but, as mentioned, it provides a stronger benchmark and can still be calculated without a great degree of difficulty, especially if one employs a computer to do so.

Figure 2 displays a plot of the probability that a network has minimum node degree two for an area of 1000mx1000m, based on the transmission range and the number of nodes. The plot consists of a plateau of high probability of connection, which drops off sharply to almost no probability. This indicates that for a given amount of nodes in an area of that size, there is a narrow range of transmission ranges for which the value will actually matter. Outside of this range, the transmission range is either too short for the distribution of nodes to be able to effectively communicate among each other, or the nodes are too close together for an increase in the range to actually matter.

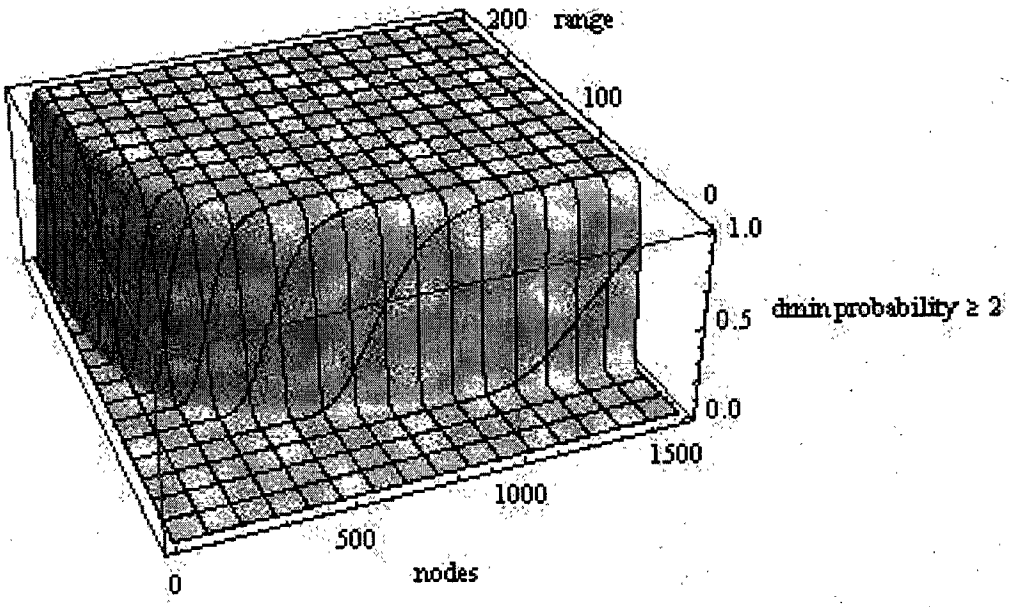

Figure 2: Plot of Equation 5 with $d_{min} = 2$.

This still leaves the question of connectivity open, which is a far more useful benchmark, since it guarantees that no "islands" will exist in the network. Because k-connectivity provides redundancy when nodes fail, it has even greater utility. To model connectivity, the author considers the links in the graph, not just the nodes. A fully random graph, with both nodes and links created at random, cannot adequately function as a model for a MANET in this case, since the links will not necessarily correspond to connections. Thus, [3] uses random geometric graphs (see Figure 3 for an example), which have random node placement and links which exist between nodes if the nodes are within a certain distance of each other. This directly models MANETs, given the paper's assumptions. Conveniently, Bettstetter uses previous findings that, given a large number of nodes, there is high probability that a random geometric graph will have k-connectivity when it has minimum node degree k. Thus, the formulas provided earlier for minimum node degree also apply for k-connectivity, when using high probabilities. In the paper, Bettstetter demonstrates that this holds true for networks with 1000 nodes distributed over a 1000mx1000m area. It should be noted that this figure uses toroidal distance–nodes near one border are considered to be close to those near the opposite border–to improve its results, which may be disingenuous since

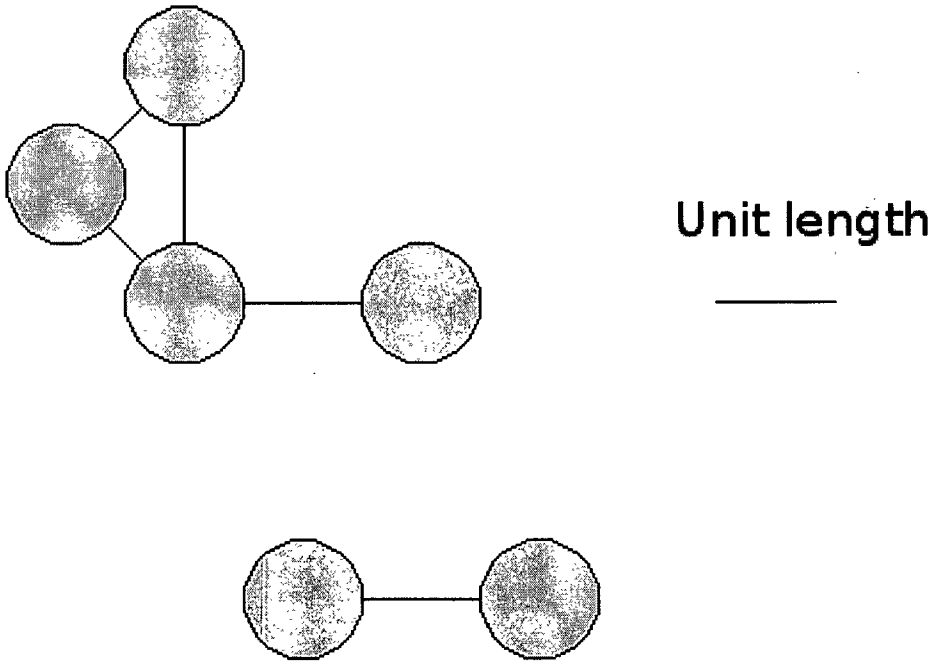

Figure 3: Random geometric graph, with distance equal to the width of a node. Note that the bottom two nodes are not connected to the upper group, since they are too far away.

it does not reflect reality.

The assumptions in this model are fairly limiting. For one, the model does not consider imperfect links or the possibility of fluctuations in signal power. While the simulated results nonetheless generally agree with those of the model, the problem cited in [11], that simulations do not account for fluctuations in signal strength, means that these omissions could still be a major issue in the model's accuracy. This, however, can be avoided by treating the range of the wireless device analyzed to be the minimum range that is likely to be encountered; since the model is probabilistic, this should not cause many problems. More problematically, the model only works under certain mobility models; specifically, those which preserve a uniform distribution of nodes. This seems weak when considering the practical applications of the model. On the other hand, taking these issues into consideration could lead to the model becoming overly complex and difficult

to analyze in a meaningful way.

Despite these objections, the model still works overall as an approximation of connectivity; however, the model cannot be exactly applied to the networks formed by this protocol. Bluetooth nodes will only be able to connect to one other Bluetooth node at any given time, regardless of the number in range, and a given Bluetooth access point can only connect to seven other devices. Still, finding the probability of a minimum degree of one and 1-connectivity for a network should give some idea of the potential performance of the network (likely an upper bound).

As mentioned earlier, the necessary node density for a minimum node degree of one given a range and a desired probability is 2. For a Bluetooth network, r_0 is approximately equal to 10 m. Assuming a network with 500 Bluetooth nodes, a desired probability of .99 will require a minimum node density of 0.034 nodes per m^2, or a minimum area of 14,700 m^2 (121 m to a side). In contrast, an 802.11g network has an outdoor range of about 140 m [36], which for the same conditions will require a minimum node density of $1.5 \cdot 10^{-4}$ nodes per m^2, or a minimum area of $3.4 \cdot 10^7 m^2$ (5830 m to a side). In addition to this, Figure 4 provides a plot of the necessary density versus the desired probability and the number of nodes. Somewhat unexpectedly, an increase in the number of nodes increases the required node density to a degree that a smaller area is required.

The node density value calculated above serves as a lower bound for this protocol. The equation assumes that any Bluetooth node will be able to connect to any other; this is not the case, as PANU nodes can only connect to NAP nodes. Given an equal distribution of NAP and PANU nodes, these numbers should reflect reality, as 1-1 connections for every device should be entirely possible. However, in the case where there are many more PANU nodes than NAP nodes, which is highly likely, the concentration of Bluetooth nodes will have to be greater.

Even assuming the best case, it is significantly more difficult to establish a high degree of connectivity with uniformly distributed Bluetooth nodes than it is with 802.11 nodes, as one would expect. Thus, Bluetooth nodes will likely have to make an effort to cluster around NAP nodes to maintain connectivity, which hurts the mobility of the network. Furthermore, NAP nodes are limited to seven connections, making the practicality of establishing connections even lower in situations where PANU nodes outnumber NAP nodes.

Given this constraint, it would be helpful to have some idea of how many NAP nodes are required for a given number of PANU nodes. Cordeiro et al. have explored a situation similar to that which arises in this protocol in [13], which describes a framework for delivering Internet access to Bluetooth devices using

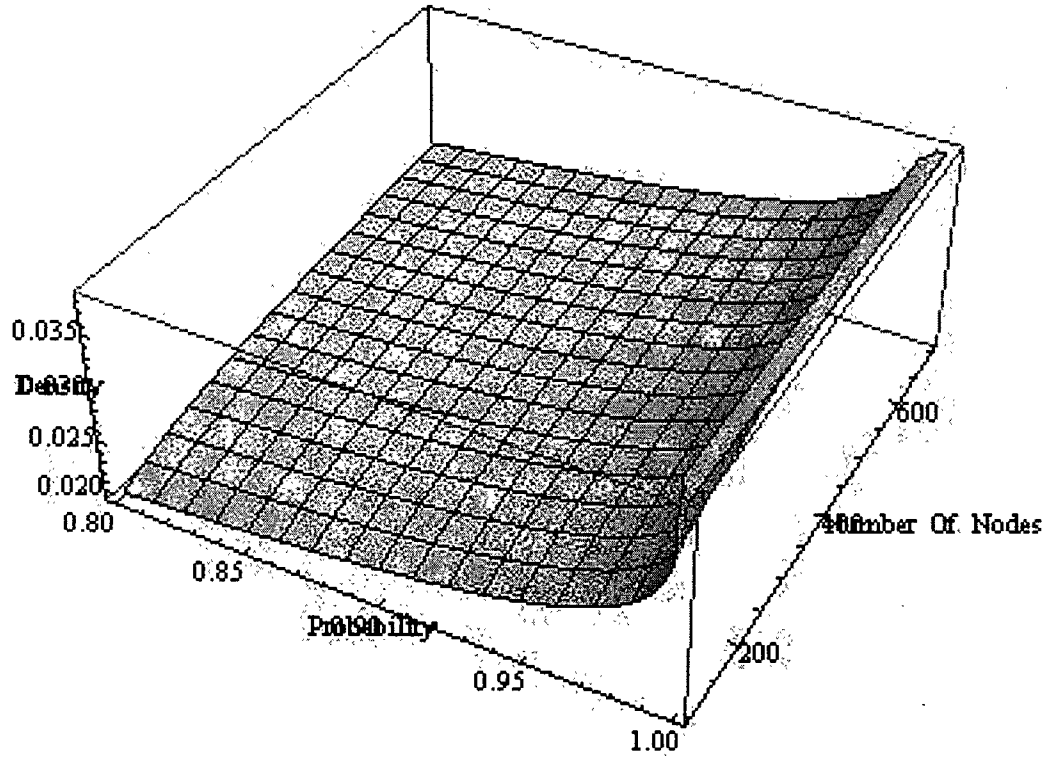

Figure 4: Plot of Equation 2 with $d_{min} = 2$.

specially designated "gateways", which function like NAP nodes. At the end of the paper, they derive an upper bound on the number of gateways needed in a network.

However, the assumptions they use are quite different from those in this paper. They assume that the master of each piconet will not be the gateway node. More importantly, their network assumes the existence of scatternets, and that each gateway will bridge two piconets. Though the authors do not explicitly state this, mobility in this network seems to be minimized, if not existent at all. Thus, their paper has limited direct applicability to this project. However, one can compare the number of gateways needed for this architecture to the ideal number of NAPs needed in my architecture to see which method yields better results in this area.

In [13], the piconets are assumed to be circles centered around the master node, and a gateway occurs at the intersection of two piconets Therefore, to determine the maximum number of gateways needed, the piconets must form a circular arrangement in which each piconet lies tangent to several other piconets. Specifically, the arrangement that they form is the most compact arrangement of circles possible; refer to the paper for more detail. The authors then use properties of circular geometry and the area covered by each piconet to derive an equation for an upper bound on the number of gateways in a network: $\lfloor \frac{7n}{2} \rfloor - 2\lfloor 4\sqrt{n} - 4 \rfloor$, where n is the number of piconets in the network. Specifically, they determine that the number of boundary piconets is $\lfloor 4\sqrt{n} - 4 \rfloor$, by finding the area of the annulus that constitutes the border piconets and dividing it by the area of a single piconet. Since each piconet can have up to seven nodes bordering on another piconet (the master cannot lie on a boundary), there can be $7n$ gateways. However, for each boundary piconet, there are four nodes that cannot border on another piconet, so one must subtract four times the number of boundary piconets. Finally, this would double-count the gateways shared between piconets, so the total must be divided by two, yielding the prior equation.

Given this equation, a network with 72 piconets (up to 500 PANU nodes) would require 188 gateways. To show that this is the maximum, the authors point out that an arrangement with fewer piconets on the boundaries will have more gateway nodes, and a circle has the shortest perimeter for a given area. This means that no arrangement can have fewer boundary piconets and therefore, no arrangement can have more gateways.

As mentioned before, this formula is mainly useful for the sake of comparison to the results of this paper. According to this, the worst-case ratio of all nodes to gateway nodes is about 2.7. If this project requires a node-to-NAP proportion

less than or close to 2.7, it will work better than, or at least as well as, BlueStar. However, BlueStar does not account for connection efficiency in their equations, looking instead for coverage, so it will likely be necessary to look at the sum of the time spent connecting and the time spent connected when simulating this paper's protocol to determine whether it performs comparably to Bluestar.

5.2.2 Information Dissemination

The ability for MANETs to communicate is not the only important factor in analyzing their behavior; efficiency in communication also plays a critical role. Generally, most analyses of MANETs employ epidemic models in determining how quickly information will spread throughout the MANET. Again, this analysis is important for Bluetooth, given both its low bandwidth rate and range.

Ref. [24] takes a fairly basic approach to such modeling. The paper employs a basic SI compartment model from infectious disease epidemiology, wherein S(t) represents the number of susceptible nodes (i.e., those looking for information) and I(t) represents the number of infected nodes (i.e., those with the information). The authors begin with the standard equations

$$\frac{dS(t)}{dt} = -\alpha S(t)$$
$$\frac{dI(t)}{dt} = \alpha S(t)$$

Here, α is a parameter that describes the diffusion of data between the "infected" and "uninfected" nodes. In this case, $\alpha = \frac{\beta x}{N} I(t)$, where β is the probability that data will be transferred when an infected node contacts an uninfected node and x is the number of contacts that a given susceptible node makes per unit of time. The authors then substitute a for the coefficient $\frac{\beta x}{N}$, describing a as "the progress of information dissemination in the MANET". They then derive the equation $I(t) = \frac{N}{1+(N-1)Ne^{-aNt}}$ using standard solving techniques for first order equations. Here, I(t) is the rate at which nodes receive information, and N is the number of nodes. Unfortunately, deriving the value of a, which depends on the MANET algorithm used, the mobility model followed by the nodes, and the characteristics of the communcation medium, among other factors, can only be accomplished through simulation, which is beyond the scope of this paper.

Using the value found for a, the authors find an optimal node density of 620km^{-2}. It should be noted that the parameters for the model appear somewhat bizarre in certain cases; the communication range is set to 75 m, and the communication rate to 2048 Kbits/s (twice that of Bluetooth), corresponding neither

to Bluetooth nor any of the iterations of 802.11. Given this, though, the optimal node density for the Bluetooth nodes in this protocol will naturally be higher.

In contrast to the fairly simple model described above, [26] uses a complex series of probability equations to analyze the performance of a buffered data sharing algorithm, seven degrees of separation, over a MANET given parameters of the devices including the transmission range, the buffer size, and the size of the area over which the devices are distributed. The primary characteristic that the model measures is the buffer hit rate, which depends on the probability of accessing the information in local buffers and in remote devices.

The model provided is quite thorough, but its applicability to my current situation is limited. As all the models discussed here do, the paper assumes a homogeneous network, wherein all devices are identical. This could be worked around; however, much of the efficiency analysis relates to buffer size and similar features, which will vary widely depending on the type of machines running the protocol. While I have only tested the implementation on full computers, it is entirely possible that handheld devices with significantly less available memory could be running stripped-down AODV implementations. Due to these issues, and because of the model's complexity, implementing it and testing it for Bluetooth networks lies beyond the scope of the paper. In addition, the math of the paper is closely tied to the buffer system used, so I shall not go into detail on it here. That said, it should be noted that, for the parameters of the equation they used (similar to 802.11g), an increase in the number of nodes generally yielded a small linear increase in performance (about 10 percentage points from 50 to 250 nodes). While it should be noted that the area considered is relatively small (1000x1000 meters), these results may indicate that, in general, once there is a decent level of connectivity within the network, increasing the number of nodes will only gradually affect the distribution of data.

5.3 Simulation Models

While analytical models are quite useful in their own right, they have their limitations. In order to be tractable, they need to make broad simplifying assumptions, such as a uniform node distribution throughout the sample area, and they often fail to account for details of the specific protocol being used. To obtain more accurate models, one must ultimately turn to simulations.

The complexity of such models often varies quite a bit. [34] uses a fairly simple simulation to back up heavy analysis work. This paper looks at a very similar issue to that explored in [3], seeking to ascertain the "critical transmission

range" given a number of nodes. Unlike [3], the range is determined using a graph that connects the nodes based on their "direct neighbors", pairs of nodes closer to each other than any other nodes. The algorithm then constructs a minimum spanning tree over this graph, and finds the longest edge, the length of which is the critical transmission range. Though there is a strong theoretical basis behind the model, it cannot be effectively used without simulating node movement using a computer, which also allows multiple mobility models to be considered.

The model used three different mobility models (random waypoint, a random Gauss-Markov model, and an unrealistic model in which nodes randomly appear at points in the area) and an area of 500x500 units. In all cases, which included up to 100 nodes, the mean critical transmission range was over 100, agreeing with the results of [3] when applied to Bluetooth MANETs. This model lacks some of the pitfalls of the analytical connectivity model, but since it requires a simulation and the experiments performed with the simulation are fairly limited in scope, it is difficult to make any conclusions about the performance of the mixed Bluetooth-wifi protocol proposed here. However, since 802.11 has a significantly longer range than Bluetooth, the possibility exists that, in many cases, critical transmission range will not be a problem, though, once again, this points to a need for the PANU nodes to cluster around the NAP nodes.

The above model is something of an exception regarding simulation models for MANETs. The majority of simulation models go into heavy detail in simulating networks, attempting to simulate nearly every aspect of the protocol and the environment. [23] provides a list of the components that generally comprise a simulation: an underlying simulation program (such as ns-2 or JiST/SWANS); a simulation of physical characteristics, like the mobility model and transmission range; emulation of the underlying transmission protocol, such as 802.11 or Bluetooth; a version of the network protocol programmed for use in the simulator; and some means to collect and analyze results, which include metrics like routing overhead, the ratio of packets delivered to those received, and the length of paths through the network [7]. The code for the network simulator that emulates the protocol ends up looking very similar to the actual code for the protocol. In AODV-UU, for instance, the simulator code and the real code share the same files; the two are distinguished using a compiler option when building the program.

The first option that confronts one when trying to create a detailed simulation such as this is which simulator to use. ns-2 seems to be by far the most commonly used simulator, and it has a good deal of libraries written for it, including emulation of Bluetooth. [23] makes a fairly strong case for JiST/SWANS, arguing in favor of its scalability, faster speed, and implementation in Java, rather than C. Al-

ternatively, given the complexity of these sort of programs, one can try to code a MANET simulation by hand; this will likely not be able to be as comprehensive as simulations using pre-existing software, but may be adequate for one's purposes.

Another highly important factor is the selection of the mobility model to use. [9] provides an in-depth analysis of how the mobility model affects the performance of MANETs, looking at both individual and group models. The paper spends the most time analyzing the Random Walk model, where nodes randomly choose directions and speeds at which to travel, and the Random Waypoint model, wherein nodes pause for a time and then choose a random destination and speed. Other models analyzed include the Gauss-Markov model mentioned earlier and the Nomadic Community Mobility model, in which nodes travel together in clusters. The paper concludes that the mobility model chosen can have a dramatic effect on performance, and recommends that the mobility model should be chosen based on the anticipated scenario. It also gives specific recommendations for models in general circumstances, particularly recommending the Random Waypoint and Gauss-Markov models.

6 Testing

6.1 Introduction

The implementation section proved that Bluetooth and 802.11 MANET integration could be done, and the modeling section gave some idea of how this would work; now, it remains to demonstrate more specifically whether it can be done effectively. In order to do this, I must perform two steps. First, I must do actual, physical testing to get an idea of the various physical parameters affecting the network. Ideally, I would then be able to perform a rigorous physical test of the MANET, but a dearth of equipment makes this impossible. Therefore, I must do simulation testing, using the parameters gleaned from the previous phase to make this as realistic as necessary. In these latter tests, I will principally look at the average time each node spends connected, as well as the time each spends connecting. The former will give some basic insight into how usable the network is; the latter will help indicate whether any problems that exist are because of Bluetooth's low range (if the time spent connecting is relatively low) or because of Bluetooth's slow connection times (if the time spent connecting is relatively high).

6.2 Preliminary Performance Testing

Before actually testing the performance of AODV with Bluetooth, I first did some basic testing of Bluetooth's capabilities to get some idea of how well Bluetooth performs in general. The first tests I did were of the time for a PANU device to discover and connect to a NAP node. I did three instances of these tests, with one, two, and three NAP nodes present. The connection times in milliseconds are as follows, along with the NAP to which the PANU connected. The first NAP was immediately next to the PANU, the second was on the floor above the NAP, and the third was in the same room. These tests strictly measure discovery and connection time; they do not account for the time it takes to discover services.

From this data, several details become apparent. First, the proximity of each NAP to the PANU seems to have little effect on the NAP to which the PANU will connect. Secondly, increasing the number of available NAPs from one to two dramatically increases connection time, though going from two to three yields little, if any change (using three NAPs yielded no outliers, but this is not likely to be significicant). This indicates that it may be better for to have a network in which the NAP distribution is relatively sparse, as a ten second connection time could cause severe problems for any application that relies on uninterrupted data

Time (ms)
10750
823
768
1127
618
857
1318
1977
1186
11478
848

Table 1: One NAP

Time (ms)	NAP
11062	1
10638	2
11420	1
11365	1
10997	1
384	2
10972	2
10890	2
10635	1
11696	1
1142	2
10908	2

Table 2: Two NAPs

transfer, like VOIP. However, it could be difficult to both achieve this and ensure that almost all PANUs have a NAP with range.

When the service check before connection is implemented, the times for a PANU to connect when there is a single NAP in range resemble those to connect to two or more NAPs. The times appear in the table below; naturally, these will be increased if there are devices in range that do not support the NAP service (like other PANUs).

Time (ms)	NAP
11561	2
11459	2
11498	3
11539	3
11955	1
10750	2
11733	1
11789	2
11521	2
12546	2
11467	2
10761	1

Table 3: Three NAPs

Connection Time (ms)
10826
11079
11329
11325
11117
11667
11089
11166
10735
11342
11420

Table 4: One NAP with service check

Interestingly, however, there is no noticeable difference between the version with the service search and that without when the number of NAPs is increased to two. This strongly implies that there is no change in connecting time when there are multiple available NAPs.

Time (ms)	NAP
11161	2
11158	1
11234	1
11461	2
12298	2
12324	2
11286	2
11366	2
11293	1
11342	1
11234	1
11274	1

Table 6: Connection Time With Two NAPs and the Service Search

Similarly, there is no significant difference when not all of the machines are valid NAPs, as seen in tables 8 and 10.

A difference does occur, however, when trying to connect to a NAP with one PANU to which it is already connected. The connection times are shown in Table 12. This difference is statistically significant, with a p value of .0028 when compared to the standard results for connecting to a single NAP when using the service search. Unfortunately, logistical obstacles prevented me from carrying out this experiment with more than one PANU; however, these results have an important signficance in the simulation results, as described in the section below. In addition, they indicate, contrary to previous impressions, that it may be better to have a denser amount of NAPs, so as to reduce the average number of PANUs connected to each NAP, as well as make it more likely for any given PANU to have at least one NAP in range.

	2 machines, 1 NAP	3 machines, 1 NAP
	Time (ms)	Time (ms)
	11199	11732
	11291	11752
	11263	11879
	11250	11747
	11497	11714
	12026	11653
	11252	11755
	11294	11723
	11239	11778
	10981	11737
	11002	11763
	10999	12434
	11207	12704
	10996	11800
	11228	11779
Average	11248.3	11863.3
Standard Deviation	258.204	294.982

Table 8: Connection Times With One Valid NAP and One Non-NAP Device

	Time (ms)	Connected NAP
	11872	1
	11918	1
	11619	2
	11683	1
	11879	1
	11883	2
	11861	1
	11820	2
	12472	2
	11824	2
	11773	2
	11915	2
	11892	1
	11886	1
	11784	2
Average	11872.07	
Standard Deviation	186.54	

Table 10: Connection Times With Two Valid NAPs and One Non-NAP Device

	One PANU already connected	No PANU connected
	Time (ms)	Time (ms)
	13848	11369
	14931	12096
	13084	11773
	12473	11387
	12276	11574
	12697	11450
	13539	13901
	13576	11483
	14369	12100
	12510	11816
	12908	11437
	12303	12658
	12590	11371
	13173	13273
	12498	11450
	12529	11465
	12521	11358
	13410	11519
	12515	11439
	14102	11405
	24309	11200
Average	13626.7142857143	11786.9
Standard Deviation	2556.79	692.70

Table 12: Connection Times For a NAP With and Without a PANU Already Connected

After connecting the PANU and NAP nodes, I then needed to test the time it takes for the two nodes to discover each other over AODV-UU. The average value for this was 544 milliseconds, with a standard deviation of about half that, a fairly small value compared to the time to connect the nodes over Bluetooth. The full results appear in Table 14.

	Time (ms)
	772
	15
	484
	239
	997
	658
	644
	588
	702
	202
	635
	61
	282
	743
	647
	534
	95
	816
	316
	559
	584
Average:	544.19
Stdev:	269.60

Table 14: Connection Times For AODV-UU over Bluetooth

Once the PANU and the NAP were connected, I then ran a ping between the two devices for 1183 packets, over the course of 1182080 milliseconds. The first

three packets failed to transmit for some reason, but after this, the average ping time was 36.147 ms, with a minimum of 9.142 ms, a maximum of 587.992 ms, and a standard deviation of 23.343 ms.

It is worth noting that I ran into a number of errors when running these tests. The first time the PANU connection program is run, it will almost inevitably fail to connect. In addition, the program will occasionally time out when trying to connect, though this seems to occur without any particular pattern. Finally, the connection will occasionally drop for no apparent reason. This is particularly noticeable when using AODV-UU. If, on the PANU, one is merely connecting with bnep0, the operating system will enter into an uninterruptible infinite loop waiting for the bnep0 interface to be freed when the connection drops, which prevents effective use of the terminal and precludes turning off the system. This is likely fixable by using a net-bridge solution similar to that used for the NAP, since the PAN0 interface will persist after the connection dies. I have yet to test this, however.

6.3 Simulation Testing

Simulating a mixed Bluetooth-wifi network proved to be incredibly problematic. Initially, I hoped to use ns-2 to gain an accurate view of how AODV-UU would perform for such a network. This requires several extensions to ns-2 in order to function properly. First, it needs an extension to simulate the Bluetooth medium. A number of these exist, the most recent of which is UCBT [1]. In addition, it also requires support for nodes with multiple interfaces to represent the NAP-wifi nodes. The only extension I could find for this is The Enhanced Network Simulator [16].

Unfortunately, these extensions were not compatible with each other. TENS is written for release 2.1b9a of ns-2, whereas UCBT requires at least version 2.26 [1, 16]. Neither will work with a version of ns-2 that supports the other. In addition, I could not find other extensions with similar functionality that would work together. Thus, due to time constraints, I chose to implement my own simulation in Java.

In the interests of avoiding complexity, I made a number of simplifying decisions in the model. First, because the area over which I ran the tests would generally be small enough that the range of the wifi would be inconsequential, I assumed that all wifi nodes would be in range of each other throughout the experiment, and only modeled the Bluetooth connections. Based on empirical testing, I assumed the connection time to be about 11 milliseconds, and modeled time

on a second by second basis. This ignores the case in which there is only one NAP in range, as the case is unlikely to occur and checking for it would significantly increase processing time. For NAPs with multiple PANU connections, I took the worst case scenario from my empirical testing earlier, and assumed that each PANU connected to a node increased the time it takes to connect by two seconds. While this may not be accurate, any differences will likely only improve the results (unless the increase in the amount of time increases with each additional PANU, in which case the results will be worse). Finally, because it is relatively small, I ignored the AODV-UU connection time.

For a mobility model, I used a simple random waypoint method, where each node selects a destination and moves towards it. Because the area is relatively small, I assumed that movement would be accomplished by walking, which should move at a relatively constant rate of .5 m/s.

The algorithm for the simulator works as follows. First, it creates an array of nodes with a set number of NAP and PANU nodes placed at random indices throughout the array. Once this is complete, it enters a loop for a number of iterations; each iteration represents the passage of a second. In each step of the loop, the simulator iterates through the node array and updates each node therein. For all nodes, this entails moving the node closer to its set destination, or, if the node is stationary, checking to see whether it is assigned a new destination. In addition, the PANU nodes also have to check their Bluetooth connection.

Checking the Bluetooth connection involves several steps. First, if the node is not currently connected to a NAP, or "searching", it iterates through the node array until it finds a NAP node that is in range. If it succeeds, it sets its state to "connecting", sets a timer to eleven ms–the amount of time for a PANU to connect to a NAP–and returns. Otherwise, it does not change its state.

If the node is currently connecting, it first checks to see if the node to which it is trying to connect is still within range. If not, it clears its timer and resets its state to "searching"; it will resume its search from the array index of the node following that to which it tried to connect. However, if the node is still in range, it checks the timer. If the timer is currently zero, it checks the NAP to ensure that it has fewer than seven connections. If so, it sets its state to "connected" and notifies the NAP node of the connection; otherwise, it resumes searching as described above. This assumes that, when a piconet is full, other devices will be refused only after completing discovery. While this may not be the case, I lacked the resources to verify it; if devices are instead immediately refused, the connection process may go significantly faster in some cases (this is addressed later).

If the PANU node is connected to a NAP, it first performs the range check,

returning to searching if out of range and notifying the NAP that it has been disconnected. Otherwise, it increments a timer that measures how long the node has been connected. Upon being disconnected, this timer gets appended to a linked list of times; the average time that a node spends connected is then calculated by taking the average of the times contained in this list, along with the current value of the connected timer.

The simulation collects a number of statistics about the experiments, which consist of 100 runs for each set of parameters. The primary statistics that I analyzed were the average time each PANU spent connected, as well as the average time that each PANU spent connecting. In addition, the simulation also reported the average number of connections for each node, the maximum and minimum amount of time any one node spent connected, and the aggregate time spent connected by all the nodes.

In total I conducted six different experiments. Three of these varied the area's size, the number of NAP nodes, and the number of PANU nodes, respectively, with a .5 chance of movement each second after a node reaches its destination. The other three varied the same parameters, but used a .05 chance of movement. Each experiment was run for ten hours. In all cases, the results were statistically significant, except where noted.

The results of the first experiment, in which the area varies and the probability of movement is .05 appear below in Figure 5. This was conducted with 10 NAP nodes and 40 PANU nodes.

As one would expect, the average connected time begins at a high percentage of the total time of the experiment, and then gradually decreases as the area of the experiment is increased. Correspondingly, the time spent connecting gradually increases. The curves seem relatively hyperbolic, though it is difficult to say whether this trend will be continued. It seems unlikely for the connecting time curve, since the last point demonstrates a slight decrease that falls just short of statistical significance with a p-value of .054.

The corresponding experiment with .5 probability of movement, shown in Figure 6, provides similar results, but with worse performance. With the exception of the 30x30 trial, each trial seems to have its total connection time decreased by about 500, which likely results from the higher mobility making it more likely for a PANU to move out of range of its NAP. The connecting times curve is shifted correspondingly higher; interestingly, it exhibits a decrease in the final point similar to the previous data, but here, this decrease is significant. The decrease likely arises from the increased area, which makes it more probable that a PANU will be completely isolated from other nodes and thus unable to begin the connecting

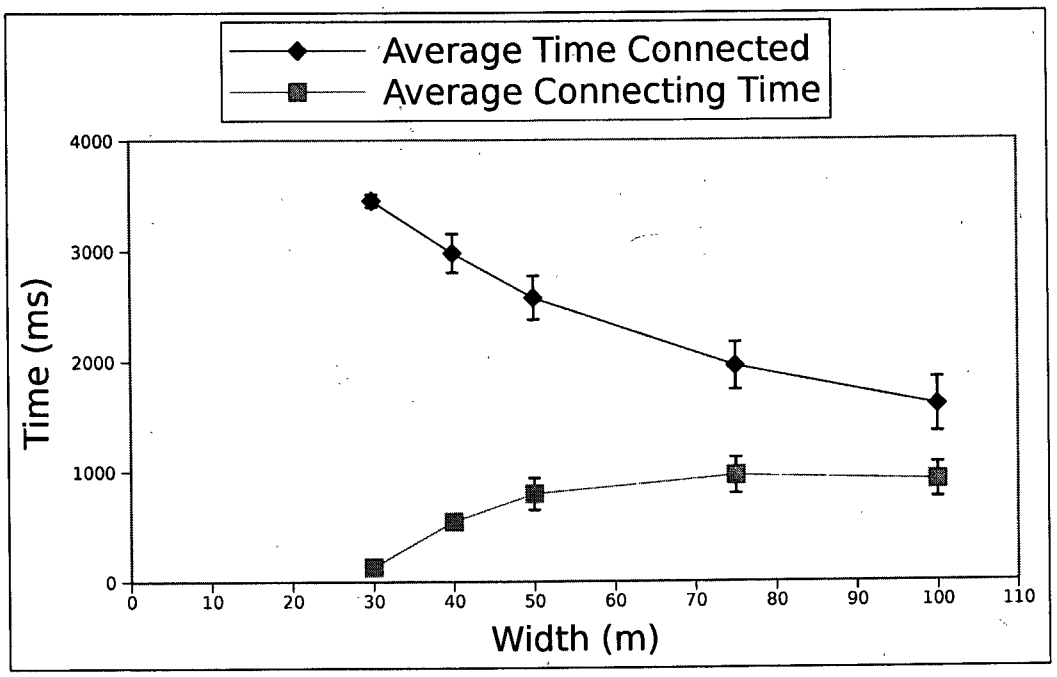

Figure 5: Varying Size, .05 Movement Probability

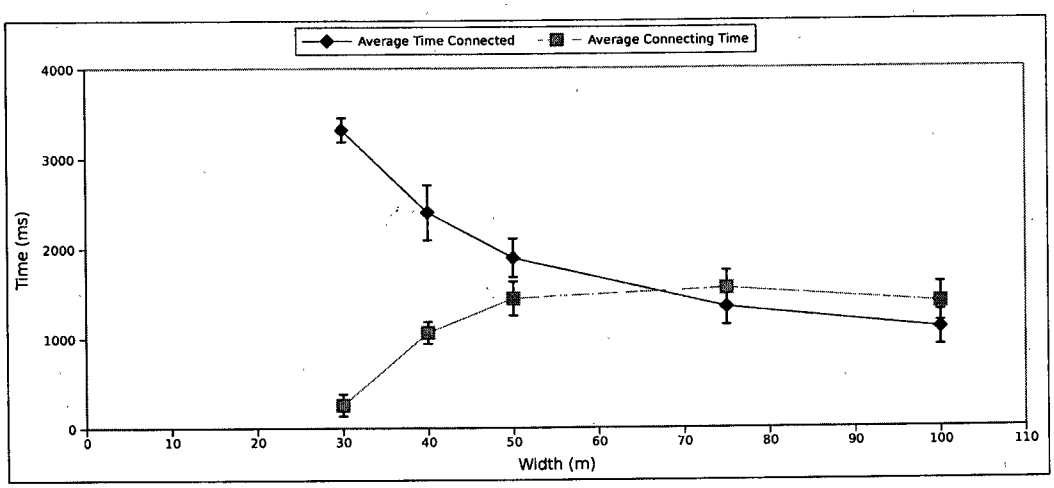

Figure 6: Varying Size, .5 Movement Probability

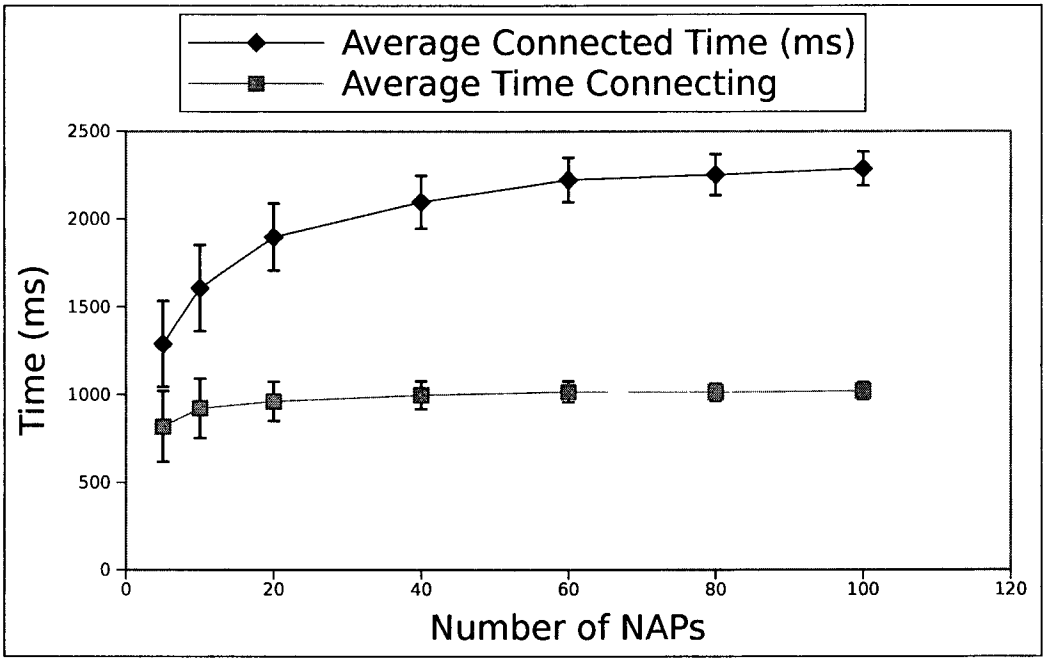

Figure 7: Varying NAPs, .05 Movement Probability

process.

Both experiments where the number of NAPs varied, shown in Figures 7 and 8, provided predictable results: as the number of NAPs increased, the average time connected increased in a roughly asymptotic fashion approaching 1800 ms for the more mobile network, and 2400 ms for the less mobile one. Again, the less mobile network outperformed the one with more mobility; the difference generally somewhere around 600 ms. In both cases, the average connecting time increased with a similar, though less regular curve, which ultimately approached 1200 ms for the less mobile network and 1800 ms for the more mobile. The curve here likely occurs because, initially, the PANU nodes have a greater opportunity to connect to NAP nodes when a NAP node is added, increasing the connecting time. Eventually, though, there are enough NAP nodes so that when one connection breaks, a node can then immediately start to connect to another, so adding additional NAPs has little effect.

The experiments with the PANUs varied also gave the expected results, shown in Figures 9 and 10, with the time connected holding an inverse linear relationship to the number of PANUs. Again, the less mobile network outperformed the more

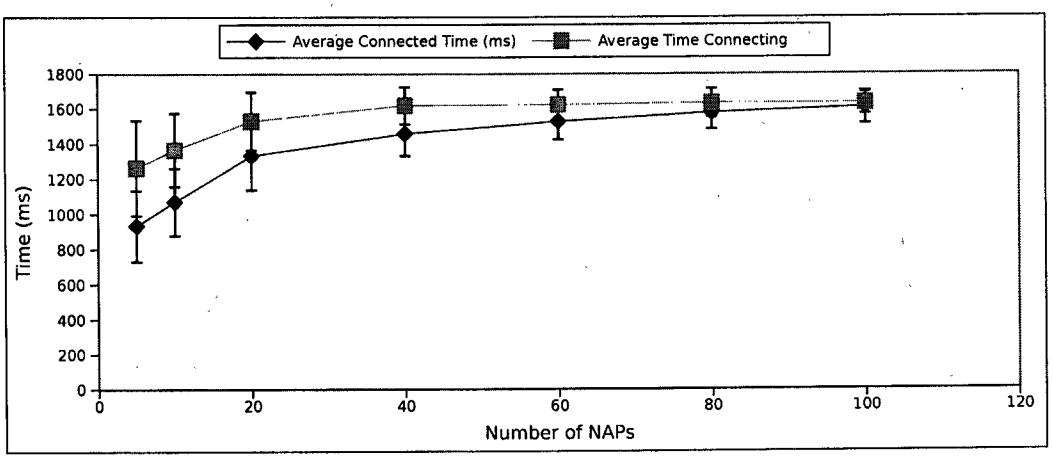

Figure 8: Varying NAPs, .5 Movement Probability

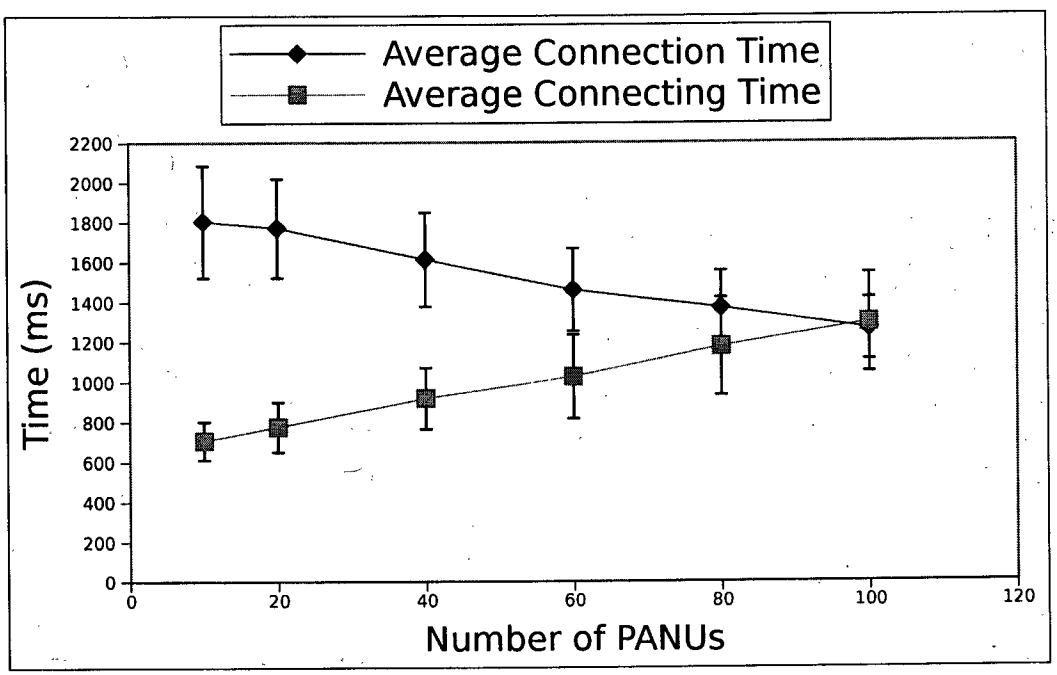

Figure 9: Varying PANUs, .05 Movement Probability

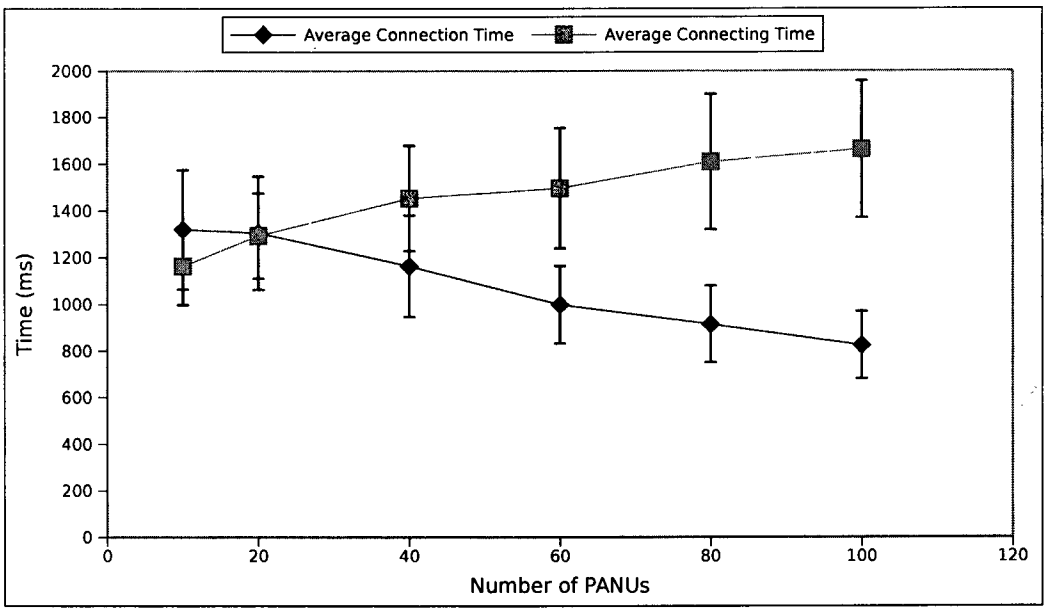

Figure 10: Varying PANUs, .5 Movement Probability

mobile one, with the connected and connecting times both differing by about five hundred for each trial. The connected time becomes greater than the connecting time significantly more quickly for the more mobile network, with the curves intersecting at 20 PANUs. In contrast, the curves for the less mobile network only intersect at 100 PANUs.

These experiments show that the most important factor in determining the performance of a Bluetooth-wifi network is the number of PANUs. Increasing the number of PANUs will have a roughly constant detrimental effect. Increasing the number of NAPs available will aid performance significantly early on, but the effect of additional NAPs will gradually grow negligible as their number grows large. This corresponds with the results mentioned in [26], which, as mentioned in section 5.3, find that adding additional nodes once the NAP is well-connected becomes ineffective. Finally, increasing the area will naturally make the performance worse, but the degree by which this does so also decreases as the area grows large.

In addition, I also tried several of these experiments with the PANUs checking the fullness of the NAP's piconet before connecting, to simulate the PANU simply ignoring these piconets, rather than trying to connect. Surprisingly, in every case tested–100x100 with 10 NAP nodes and 40, 80, and 100 PANUs–

statistically significant differences were not achieved in connected times, though they were in times spent connecting. This is likely because full piconets only become problematic when the number of PANUs significantly exceeds that of the NAPs. Using higher number of PANUs would likely cause a statistically significant difference, as the p value progressively grew smaller (.08 for the experiments with 100 PANUs), but the prospect of PANUs outnumbering the NAPs by more than ten to one seems improbable.

Finally, I also ran an experiment with 400 PANUs and 100 NAPs in a 130mx130m area with .05 movement probability. According to the connectivity model examined in 5.2.1, a network with 500 Bluetooth nodes in an area of this size should be connected close to 99% of the time (the exact area should be around 121mx121m for this probability). This, however, was not the case: on average, nodes were connected for 1568.03 seconds out of 3600, or 43.6% of the time, with a standard deviation of 85.14 seconds. The time spent connecting was almost equal to this, at an average of 1583.87 s with a standard deviation of 96.06 seconds. Thus, the model provided a good predictor of the amount of time the nodes would be in contact with each other; however, it failed to account for connection times. These connection times caused the PANUs to be connected for drastically less time than they otherwise would have been, indicating that this will be a serious factor when considering Bluetooth MANETs. In addition, that fact that the two times were almost equal indicates that the PANUs are likely seldom in contact with a single NAP for more than ten seconds at a time, since that is the amount of time it takes to connect.

7 Results

To some extent, the results of these experiments surprised me with how well they performed; I had expected something virtually unusable, but instead, the PANUs tended to be connected about one third of the time even in the worst case tested. However, this is far from ideal, as the unstable connection will cause problems for applications that require constant data transfer. VOIP would be completely infeasible, as would anything requiring streaming media.

Currently, these results will support, albeit with a good amount of potential user frustration, basic web browsing and messaging services. The latter would likely require some kind of buffering of messages on the NAP end, since it is entirely possible that the destination PANU would move out of range before the message could be completely transmitted. The network could not simply drop the

messages, as this could lead to uncertainty as to whether the message had been transmitted.

The main problem for usability, moreso than the range, seems to be the ten second or more connection time. If the only issue were range, there still would be problems with achieving a necessary density to have all nodes connected at all times. However, in situations where the density was less than this, being disconnected would not necessarily be terribly problematic, as one could hypothetically move around a bit until one came in range of another NAP. Unfortunately, though, with the long connection times, even if there is a high enough node density to hypothetically support 99% connectivity, a PANU would still encounter a delay of at least ten seconds when the NAP to which it was connected inevitably moved out of range. This kind of delay is usually unacceptable to most users.

Furthermore, the connection problem becomes more severe when one notes that the simulation only accounted for the perfect scenario: no obstructed range, no arbitrary breaks in the connection, and no failures to connect. Any number of these factors could prove problematic, again because of the lengthy connection time. I was unable to accurately measure the frequency of these occurrences, but if they occur often, they would provide yet another barrier to successfully implementing a useful Bluetooth-wifi MANET.

Admittedly, however, the current method for connecting to devices may not be not the most efficient one. The task that appears to occupy the majority of the time is device discovery. If the Bluetooth address of a device is already known, it usually takes a negligible amount of time to connect (at least when the device is the first one to connect). Thus, if there is some means of getting the Bluetooth addresses of the NAPs in the network to a PANU upon connection, connection and discovery time could hypothetically be shortened by quite a bit. A protocol for doing this would take a fair bit of investigation; in addition, I am unsure of how long it takes for an attempted BNEP connection to time out. If this time is fairly long, such address sharing might not actually save any time.

Another alternative would be to keep the NAP nodes stationary in a sort of wireless mesh network set-up. This has its own set of problems, though, mainly stemming from the eight node size limit for piconets. With this, if there are a decent number of PANU users, it is not at all unlikely that they would congregate in one place in the actual network, rather than being randomly distributed, causing the NAP in that area to fill up rather quickly. Even allowing for the use of scatternets, a large number of connections would quickly sap bandwidth from the NAP, as mentioned earlier in this paper. One potential solution would be to use multiple NAPs to cover an area, and, for low mobility networks, this is probably the best

solution.

It is worth noting that the network architecture proposed here ignores the possibility of scatternets. Scatternets could, hypothetically, improve the situation a fair bit, especially if they were used to "daisy-chain" several PANUs to a NAP; this would alleviate the problem of limited range and allow for a much smaller NAP to PANU ratio. Redundant connections created with scatternets would also make a connection break to a NAP less cataclysmic, since a PANU could have access to multiple NAPs at once. Unfortunately, support among Bluetooth devices for scatternets seems spotty and possibly non-existent. The Bluetooth adapters that I used in the course of my research claim to support scatternets, but I have no idea how this works and was unable to test this due to a lack of resources. This also would require significantly more complexity in the routing protocol, which could lead to difficulties with the small processing power found in many Bluetooth-only devices. Finally, as mentioned earlier, it is entirely possible that scatternets could introduce significant delays of their own [28].

Even with the current network architecture, the routing protocol could be problematic, as others have addressed [22]. Not only are memory-usage and processing power an issue, but actually implementing the protocol would be necessary, since current implementations of most MANET protocols are written for major operating systems like Windows and Linux. One possible means of dealing with both problems would be to write a client tailored specifically to this network architecture. Because AODV does not require the route to go out with the package, PANU nodes would only need to store the IP address of the NAP to which they are connected. Similarly, much of the protocol's instructions for dealing with things like link repair and error messages could be ignored, since all routes from the PANU go through the NAP. This should simplify the behavior of the protocol significantly. Again, though, with scatternets, a full implementation of AODV or a similar protocol would be necessary since a node could have multiple routes leading from it.

On the whole, a number of obstacles must be overcome before an integrated Bluetooth-wifi mobile ad-hoc network could become truly practical. While there is more promise than I initially thought, overall performance is still mediocre at best, and the ten second connection time could prove hugely problematic, even in well-covered networks. Given the increasing availability of wifi capabilities on small devices, barring any kind of major improvement in Bluetooth, the utility of pursuing this line of research further is debatable. That said, if some of the approaches suggested here, such as storing the device addresses, work, there could be promise.

References

[1] Dr. Dharma Agrawal and Qihe Wang. Ucbt – bluetooth extension for ns2 at univ. of cincinnati.

[2] Apple. *The Mac OS X Bluetooth Profiles and Applications*, 2007.

[3] Christian Bettstetter. On the minimum node degree and coonnectivity of a wireless multihop network. In *MobiHoc '02: Proceedings of the 3rd ACM international symposium on Mobile ad hoc networking & computing*, pages 80–91, New York, NY, USA, 2002. ACM.

[4] Bluetooth SIG. *Personal Area Networking Profile*, 2007.

[5] Bluetooth SIG. *Profiles Overview*, 2007.

[6] bluez.org. *BlueZ Wiki*, 2007.

[7] Josh Broch, David A. Maltz, David B. Johnson, Yih-Chun Hu, and Jorjeta Jetcheva. A performance comparison of multi-hop wireless ad hoc network routing protocols. In *MobiCom '98: Proceedings of the 4th annual ACM/IEEE international conference on Mobile computing and networking*, pages 85–97, New York, NY, USA, 1998. ACM.

[8] H. Cai and D. Y. Eun. Crossing over the bounded domain: from exponential to power-law inter-meeting time in manet. In *Proceedings of the 13th annual ACM international conference on Mobile computing and networking*, pages 159–170. International Conference on Mobile Computing and Networking, 2007.

[9] T. Camp, J. Boleng, and V. Davies. A survey of mobility models for ad hoc network research. *Wireless Communications and Mobile Computing (WCMC):*, 2(5):483–502, 2002.

[10] W. Chan, J. Chen, P. Lin, and K. Yen. Quality-of-service in ip services over bluetooth ad-hoc networks. *Mobile Networks and Applications*, 8(6):699–709, December 2003.

[11] Kwan-Wu Chin, John Judge, Aidan Williams, and Roger Kermode. Implementation experience with manet routing protocols. *SIGCOMM Comput. Commun. Rev.*, 32(5):49–59, 2002.

[12] T. Clausen and P. Jacquet. *Optimized Link State Routing Protocol.* IETF, 2003.

[13] C. De M. Cordeiro, S. Abhyankar, R. Toshiwal, and D. P. Agrawal. Bluestar: enabling efficient integration between bluetooth wpans and ieee 802.11 wlans. *Mobile Networks and Applications*, 9(4):409–422, August 2004.

[14] S. Corson and J. Macker. *Mobile Ad hoc Networking (MANET): Routing Protocol Performance Issues and Evaluation Considerations.* IETF, 1999.

[15] D. Cox. *Point processes.* Chapman and Hall, London New York, 1980.

[16] Indian Institute of Technology Department of Computer Science & Engineering. The enhanced network simulator.

[17] M. Dideles. Bluetooth: a technical overview. *Crossroads*, 9(4):11–18, June 2003.

[18] IETF. Mobile ad-hoc networks (manet) charter.

[19] Michael J. Jipping and Gary Lewandowski. Parallel processing over mobile ad hoc networks of handheld machines. In *MobiHoc '01: Proceedings of the 2nd ACM international symposium on Mobile ad hoc networking & computing*, pages 267–270, New York, NY, USA, 2001. ACM.

[20] D. Johnson, Y. Hu, and D. Maltz. *The Dynamic Source Routing Protocol (DSR) for Mobile Ad Hoc Networks for IPv4.* IETF, 2007.

[21] R. Kapoor, M. Kazantzidis, M. Gerla, and P. Johansson. Multimedia support over bluetooth piconets. In *Proceedings of the first workshop on Wireless mobile internet*, pages 50–55. Wireless Mobile Internet, 2001.

[22] Frank Kargl, Stefan Ribhegge, Stefan Schlott, and Michael Weber. Bluetooth-based ad-hoc networks for voice transmission. In *HICSS '03: Proceedings of the 36th Annual Hawaii International Conference on System Sciences (HICSS'03) - Track 9*, page 314.1, Washington, DC, USA, 2003. IEEE Computer Society.

[23] Frank Kargl and Elmar Schoch. Simulation of manets: a qualitative comparison between jist/swans and ns-2. In *Proceedings of the 1st international workshop on System evaluation for mobile platforms*, pages 41–46. International Conference On Mobile Systems, Applications And Services, 2007.

[24] Abdelmajid Khelil, Christian Becker, Jing Tian, and Kurt Rothermel. An epidemic model for information diffusion in manets. In *MSWiM '02: Proceedings of the 5th ACM international workshop on Modeling analysis and simulation of wireless and mobile systems*, pages 54–60, New York, NY, USA, 2002. ACM.

[25] Pradeep Kyasanur and Nitin H. Vaidya. Routing and link-layer protocols for multi-channel multi-interface ad hoc wireless networks. *ACM SIGMOBILE Mobile Computing and Communications Review*, 10(1):31–43, January 2006.

[26] Christoph Lindemann and Oliver P. Waldhorst. Modeling epidemic information dissemination on mobile devices with finite buffers. In *SIGMETRICS '05: Proceedings of the 2005 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 121–132, New York, NY, USA, 2005. ACM.

[27] S. A. Mahmud, Shahbaz Khan, Shoaib Khan, and H. Al-Raweshidy. A comparison of manets and wmns: commercial feasibility of wireless networks and manets. In *Proceedings of the 1st international conference on Access networks*, volume 267. ACM International Conference Proceeding Series, 2006.

[28] V. B. Misic and J. Misic. Performance of bluetooth bridges in scatternets with limited service scheduling. *Mobile Networks and Applications*, 9(1):73–87, February 2004.

[29] Guangyu Pei, Mario Gerla, and Tsu-Wei Chen. Fisheye state routing: A routing scheme for ad hoc wireless networks. In *ICC (1)*, pages 70–74, 2000.

[30] C. Perkins, E. Belding-Royer, and S. Das. *Ad hoc On-Demand Distance Vector (AODV) Routing*. IETF, 2003.

[31] C. E. Perkins, E. M. Royer, S. R. Das, and M. K. Marina. Performance comparison of two on-demand routing protocols for ad hoc networks. *IEEE Personal Communications*, pages 16–28, February 2001.

[32] A. Racz, G. Miklos, F. Kubinszky, and A. Valko. A pseudo random coordinated scheduling algorithm for bluetooth scatternets. In *Proceedings of*

the 2nd ACM international symposium on Mobile ad hoc networking and computing, pages 193–203. International Symposium on Mobile Ad Hoc Networking & Computing, 2001.

[33] Michael Schmidt. Howto set up common pan scenarios with bluez's integrated pan support.

[34] Miguel Sánchez, Pietro Manzoni, and Zygmunt J. Haas. Determination of critical transmission range in ad-hoc networks.

[35] C.-K. Toh, Richard Chen, Minar Delwar, and Donald Allen. Experimenting with an ad hoc wireless network on campus: insights and experiences. *ACM SIGMETRICS Performance Evaluation Review*, 28(3):21–29, December 2000.

[36] Wikipedia. Ieee 802.11.

[37] Wikipedia. List of ad-hoc routing protocols.